.

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Implementação e avaliação de algoritmos sequencial e paralelo para descoberta de rotas ótimas para veículos

Trabalho de Conclusão de Curso

Jackson Tavares da Costa

Departamento de Computação/UFS

São Cristóvão – Sergipe

2019

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Jackson Tavares da Costa

Implementação e avaliação de algoritmos sequencial e paralelo para descoberta de rotas ótimas para veículos

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Ricardo José Paiva de Britto Salgueiro
Coorientador(a): Thiago Xavier Rocha de Souza

São Cristóvão – Sergipe

2019

À minha mãe Noélia Tavares da Costa, minha família e às memórias de meu Pai José Hunaldo da Costa, tia Telia Tavares dos Santos Silva, Avô Helvecio Tavares dos Santos, Avó Ana Pereira dos Santos e todos aqueles que de alguma forma se fizeram presente nesse caminhar.

Agradecimentos

Em primero lugar, agradeço a Deus por estar sempre comigo nessa jornada.

À Érica Paloma por ter me mostrado esse caminho acadêmico. Um universo de conhecimento que possibilita a real liberdade, pois de fato o conhecimento liberta.

Agradacer a minha mãe Noélia Tavares da Costa e meu irmãos Andréia Tavares da Costa, Jordão Tavares da Costa e Hércules Tavares da Costa pelo incondicional apoio ao vir cursar, se fazendo sempre presentes. Ao amigo Pr. Mailton e toda família, por terem me acolhido logo quando cheguei em São Cristóvão. À eles, minha eterna gratidão.

Agradecer especialmente ao professor Ricardo Salgueiro por ter aceitado ser meu orientador não só deste trabalho de conclusão de curso, mas também de iniciações ciêntíficas e, ao também professor Thiago Xavier por ter aceitado ser o meu coorientador deste trabalho como de todas as iniciações ciêntíficas quais participei, sempre disposto a ajudar. Agradecer também a todos os professores do departamento de computação e demais departamentos, por terem passado um pouco do conhecimento e sempre estarem de forma direta ou indireta incentivando a superar nossos limites.

Aos meus amigos de mais de 15 anos, Luiz Adriano, Evelyn Costa e Roseli Ribeiro. Mesmo apesar da distância e dos desencontros constantes, sempre se fizeram presente, com boas risadas, histórias e uma relação sempre amistosa.

Aos caros amigos de curso, como Davi Silva, Patrick Jones, Davisson Ednei, Gabriel Guedes, João Manoel, Ualas Lima, Murilo Formiga, Adrian Costa, Elias Rabelo, Hugo Barreto, Alana Lucia, Jusley Arley, João Matheus, Paulo de Brito, Italo Jorge, Pedro Henrique, Antonio Carlos, Diogo Lima, Werthen de Castro, Tiago Conceição, Gabriel Leite, Itauan Eduão.

Aos bons amigos também, que não são do mesmo curso como Felipe Rocha, Romário Bispo, Jonathan Santos, Fillipe Paz, Elder Cleiton, Renata Lopes, Leilton dos Santos, Sr. Manoel, Leonardo Ikejiri.

Para nós os grandes homens
não são aqueles que resolveram os problemas,
mas aqueles que os descobriram.
(Albert Schweitzer)

Resumo

O aumento significativo da população nos grandes centros urbanos, criou algumas problemáticas analisadas e tratadas nesse trabalho como a mobilidade urbana e sua eficiência, devido a necessidade cada vez maior de uma locomoção rápida e de baixo custo. Nesse cenário, as *Smart Cities* (Cidades Inteligentes) disponibilizam uma variedade de soluções que possibilitam reduzir tais problemas, gerando soluções diversas e de maneira sustentável. A partir disso, realizou-se uma simulação de tráfego de veículos em regiões da cidade de Aracaju no estado de Sergipe - Brasil e em Manhattan no estado de Nova Iorque - EUA, na ferramenta de simulação de tráfego SUMO *(Simulation of Urban MObility*), onde se deu a execução do algoritmo A-Star clássico na literatura, implementado de forma paralela, utilizando CUDA da NVIDIA e uma GPU (*Graphics Processing Unit*) que por sua vez, realiza cálculos complexos de forma eficiente quando comparada a CPU (*Central Process Unit*) e também implementado de forma sequencial, que executa uma instrução por vez na CPU. A utilização de GPU em si, além de vantajosa devido ao seu poder de realizar operação de ponto flutuante, é também uma solução de dispositivo viável pelo seu baixo custo financeiro, evitando assim uma infraestrutura de supercomputadores para realizar tal tarefa. Dessa forma, o algoritmo implementado de forma paralela neste trabalho e executado na GPU, gerou resultados satisfatórios quando comparados aos algoritmos implementados sequenciais. Esses resultados obtidos, nos mostram o poder e eficiência do paralelismo ao gerar soluções rápidas e de forma eficiente. Os resultados demonstram que é possível vislumbrar a execução dessa solução em *Smart Cities* para milhares de veículos, ao propor algoritmos de baixo custo para a definição de rotas otimizadas no ambito do algoritmo *A-Star* e o Dijkstra, podendo dessa forma, substituir toda uma infraestrutura de supercomputadores por GPUs.

Palavras-chave: GPU, CUDA, Paralelismo, Smart Cites, Rotas.

Abstract

The significant increase in the population in large urban centers has created some problems analyzed and addressed in this work as urban mobility and its efficiency, due to the increasing need for fast and low cost mobility. In this scenario, Smart Cities offers a variety of solutions that enable you to reduce all problems, generating diverse and sustainable solutions. From this, a vehicle traffic simulation was performed in regions of the city of Aracaju in the state of Sergipe - Brazil and in Manhattan in the state of New York - USA, in the traffic simulation tool SUMO (Simulation of Urban MObility), where the classic in the literature A-Star algorithm was executed, implemented in parallel, using NVIDIA's CUDA and a Graphics Processing Unit (GPU) which in turn performs complex calculations efficiently when compared to the CPU (Central Process Unit) and also implemented sequentially, which executes one instruction at a time on the CPU. The use of GPU itself, as well as being advantageous due to its power to perform floating point operation, is also a viable device solution for its low financial cost, thus avoiding a supercomputer infrastructure to accomplish such a task. Thus, the algorithm implemented in parallel in this work and executed in the GPU, generated satisfactory results when compared to the implemented sequential algorithms. These results show us the power and efficiency of parallelism to generate fast and efficient solutions. The results show that it is possible to envision the implementation of this solution in Smart Cities for thousands of vehicles, by proposing low cost algorithms for the definition of optimized routes within the scope of the A-Star algorithm and Dijkstra, thus replacing an entire infrastructure. of supercomputers by GPUs.

Keywords: GPU, CUDA, Parallelism, Smart Cites, Routes.

Lista de ilustrações

Lista de abreviaturas e siglas

BFS	Best-First Search
CPU	Central Process Unit
CUDA	Compute Unified Device Architecture
DLR	Deutsches Zentrum für Luft- und Raumfahrt
ELAN	Experimental Laboratory in computer Networks
GPU	Graphics Processing Unit
GPS	Global Positioning System
HPC	High Performance Computation
IBGE	Instituto Brasileiro de Geografia e Estatística
LCAD	Laboratório de Computação de Alto Desempenho
SIT	Sistemas Inteligentes de Transportes
SUMO	Simulation of Urban MObility
VANET	Vehicular Ad hoc Networks
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Sumário

1 Introdução

No mundo, a população que mora em cidades é 54%, podendo alcançar 65% no ano de 2040 (POSTSCAPES, 2015). Há algumas décadas, o Brasil fez este tipo de transição e de acordo com o IBGE, foi observado pelo Senso de 2010 de que os centros urbanos contabilizavam cerca de 84% do total da população (IBGE, 2010). Os centros urbanos tem recebido transições de grande parte da população mundial que viviam em regiões rurais, ocasionando em implicações na forma como as cidades são gerenciadas e organizadas.

A grande concentração de pessoas nos centros urbanos ocasiona em perda na qualidade dos serviços que devem ser oferecidos por esses centros, como sistema de saúde, segurança, mobilidade urbana, infraestruturas apropriadas, entre outros (WEISS; BERNARDES; CONSONI, 2013). COSTA (2014) diz que, uma mobilidade urbana ruim prejudica a economia e a qualidade de vida devido aos acidentes que na maioria das vezes tornam o cidadão dependente do Estado afetando diretamente a previdência, como também a perda de tempo ocasionada pelo deslocamento do trabalho para casa e vice-versa, além do desempenho dos funcionários devido aos atrasos e estresse vivenciados no transito. Com isso, é cada vez mais necessário um melhor planejamento de rotas.

A mobilidade urbana não é uma preocupação única e exclusiva do Brasil ou de países em desenvolvimento. Muitas propostas, utilizando tecnologias ou não para solucionar tais dificuldades estão sendo adotadas em outros países em busca de uma mobilidade cada vez melhor e padronizada (COSTA, 2015). Alguma melhoria nesse sentido pode ser obtida através de meios eletrônicos como GPS (*Global Positioning System*) e físico como vias interconectadas por uma infraestrutura de rede, auxiliando na tomada de decisão, como melhor rota (caminho) de locomoção de um ponto de origem até um ponto de destino.

Entretanto não é uma tarefa simples, pois é necessário que o roteamento de tráfego seja feito em um tempo cada vez menor, de forma rápida e eficiente para que o usuário possa efetuar o seu deslocamento sem maiores problemas (BARTH; BORIBOONSOMSIN, 2008).

Paralelamente a esses desafios, existe um avanço significativo da Tecnologia da Informação e de sua aplicação em atividades rotineiras, o que torna promissora e atrativa a criação de *Smart Cities* nesse cenário (TOWNSEND, 2013).

As *Smart Cities* teve seu conceito declarado pela primeira vez após a ocorrência do movimento que ficou conhecido por crescimento inteligente, na década de 90, onde a criação e implantação de políticas urbanas inovadoras foi defendida (WEISS; BERNARDES; CONSONI, 2013). A relação entre o crescimento das cidades e seu fluxo de dados massivo e sua população possibilita a criação de um laboratório cívico do ambiente urbano, onde a tecnologia é ajustada para atender as demandas específicas da região (TOWNSEND, 2013).

A necessidade de que as cidades sejam inteligentes na maneira com a qual administra seus recursos, sua infraestrutura e serviços oferecidos à população, tem sido cada vez mais almejada (Naphade et al., 2011). Em POSTSCAPES (2015), mostra que as cidades estão buscando cada vez mais utilizar diversos tipos de tecnologias combinadas no intuito de aumentar sua eficiência, reduzir custos e prover maior qualidade de vida aos seus cidadãos. Tudo isso como resultado da junção de: a) acesso à comunicação, principalmente redes sem fio; b) sensores de potência baixa com custo baixo; c) Uso de aplicativos para alcançar o engajamento da população.

A tecnologia é utilizada como ferramenta pelas *Smart Cities* para que seus cidadãos tenham uma melhora na qualidade de vida, fazendo uso de um sistema de informação muito consistente e conectado para que os recursos de determinada cidade tenham uma boa otimização e a partir do cruzamento de informações obtidas de vários setores administrativos, conseguir gerenciar de forma contínua as suas demandas.

Com isso, a redução do tempo de deslocamento é de interesse de pesquisadores e desenvolvedores no tocante a cidades inteligentes, que além do descrito acima, Shimoura et al. (1995) também constituiu seu conceito por uma das dimensões de aplicações do SIT (Sistemas Inteligentes de Transportes), que possibilita a colaboração em soluções em que a economia no consumo de combustível e diminuição de congestionamentos, reduzindo os tempos de descolamento nas viagens e na prevenção de acidentes.

Nesse sentido, esse trabalho visa implementar algoritmos que proponham rotas de tráfego de veículos, de maneira eficiente e eficaz, devido a necessidade de serviços de atendimentos cada vez mais rápidos das viaturas de polícia, ambulâncias, deslocamentos, entre outros, melhorando a qualidade de vida da população de maneira geral. Para tal proposta, as cidades precisariam de uma infraestrutura interligadas em postes, carros e semáforos, conectados e operando de forma conjunta, gerando informações e dados para auxiliar na tomada de decisão, para que se obtenha a melhor rota possível até o destino que se deseja.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo desse trabalho de conclusão de curso é implementar um algoritmo sequencial e outro paralelizado, que terão seus resultados analisados em busca do melhor desempenho para que a cidade inteligente o tenha integralizado a um sistema de gerenciamento de tráfego. Visando possibilitar assim, um planejamento de caminhos otimizados para vários veículos de determinada região, resultando dessa maneira em uma solução ótima e de baixo custo computacional.

1.1.2 Objetivos Específicos

Para que seja alcançado o objetivo desse trabalho, alguns objetivos específicos são essenciais serem , tais como: (i) Encontrar e estudar uma ferramenta computacional de simulação de tráfego; (ii) Encontrar e analisar um algoritmo de busca em grafos; (iii) Implementar algoritmos sequencial e paralelizado para a aplicação dos dados obtidos; (iv) Aplicar os dados coletados da ferramenta de simulação nos algoritmos sequencial e paralelizado; (v) Validar os algoritmos na ferramenta de simulação; (vi) analisar o desempenho; (vii) Analisar os resultados dos algoritmos elaborados.

1.2 Trabalhos Relacionados

Os trabalhos relacionados à este trabalho serão abordados nesta seção, a fim de observar e compreender as problemáticas similares e os diversos tipos de iniciativas que abordam as mais variadas soluções.

Li, He e Du (2018) desenvolveram como solução para se evitar congestionamentos, o planejando de rotas através de diagrama de Voronoi que tem-se regiões obtidas da divisão do mapa da cidade e em que os veículos tem o sistema de comunicação modelado. Métodos preditivos das condições de trânsito desenvolvidos, que utilizam um modelo de fluxo de atributos como velocidade e validam utilizando dados reais de tráfego juntamente com ferramenta SUMO (*Simulation of Urban MObility*) (HE; CAO; LI, 2012).

Wegener et al. (2006) desenvolveram um sistema descentralizado de monitoramento de tráfego com base em VANETs (*Vehicular Ad hoc Networks*). As informações coletadas pelos veículos, que extraem dados significativos gerando relatórios de tráfego. Um modelo formal de um protocolo distribuído de controle de tráfego tendo segurança como parte de seus atributos no roteamento também já foi apresentado, garantindo estabilidade após alguma falha de célula de planejamento (JOHNSON; MITRA, 2015).

Existem ainda trabalhos com foco no método ou algoritmo de planejamento de rotas, outros consideram planejamento de rota como estudo de caso, propondo métodos analisados

de disseminação de informações a partir dos nós do sistema. Em (XU et al., 2015), tem-se implementado, com base no estado atual das vias, o método de planejamento de rotas além de técnicas avaliadas em que as informações são espalhadas com o uso da VANET.

1.3 Estrutura do Documento

A organização desse trabalho tem o seguinte formato. O Capítulo 2 descreve toda a parte de fundamentação teórica, onde se dá todos os conceitos necessários para entendimento do projeto, como estudo da problemática, algoritmos a serem implementados, ferramenta de simulação, modelagem, linguagem de programação, GPU e paralelismo. O capítulo 3 trata sobre a proposta desse trabalho, a metodologia seguida e apresentação dos resultados obtidos das simulações pós implementações dos algoritmos e por fim o capítulo 4 aque abordará a conclusão do trabalho, onde os resultados obtidos serão analisados e avaliados.

2

Fundamentação Teórica

Para que seja possível a realização desse trabalho, é necessário entender toda a problemática que envolve os grandes centros urbanos com relação ao gerenciamento dos seus recursos, o aparato tecnológico das *smart cities* para auxiliar nas interconexões dos seus dispositivos e também como todo esse cenário pode ser simulado em uma ferramenta computacional. Além disso, analisar como pode-se aplicar um algoritmo em tais cenários.

A implementação do algoritmo, nesse caso o A-*Star*, necessita também de um conhecimento teórico do funcionamento do mesmo, suas características, dos prós e contras para que a solução proposta tenha, de fato, bom desempenho e seja viável a sua integração a ferramenta de gerenciamento do tráfego.

Assim, são apresentadas algumas definições de suma importância para a elaboração da aplicação do trabalho, as ferramentas necessárias para o desenvolvimento e a metodologia seguida.

2.1 SUMO

O SUMO teve seu desenvolvimento do pacote de simulação iniciado pelo DLR (Centro Aeroespacial Alemão) em 2001. Evoluiu a partir de então, incluindo uma rede rodoviária capaz de ler de fontes com diferentes formatos, utilitários de roteamento de diversas fontes de entrada, capaz de fazer simulações de alto desempenho para cidades inteiras, como também simples junções únicas (KRAJZEWICZ, 2010).

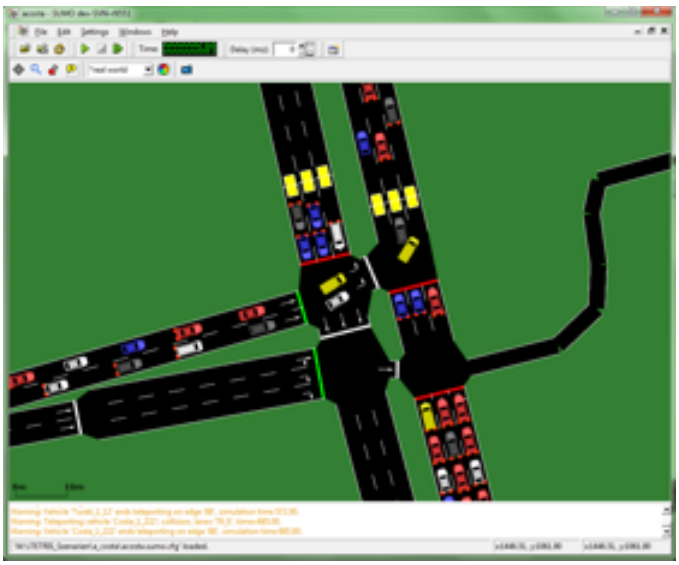
É uma ferramenta de execução de simulação de tráfego e de código aberto, que irá ajudar na análise da escolha do melhor caminho (rota) para determinado veículo. Desse modo, a ferramenta é utilizada em muitos projetos para simulação no intuito de ajudar nas estratégias a serem adotadas na gestão de tráfego. Com uma simulação microscópica, cada veículo é definido por um identificador único, a rede, a rota do veículo e o horário de partida e chegada, como a via

a ser utilizada, velocidade e a posição, tudo isso pode ser definido (BEHRISCH et al., 2011).

Nele, cada veículo pode ter suas propriedades físicas e variações de movimentos atribuídas (BEHRISCH et al., 2011). Inclui ainda ferramentas para importar redes rodoviárias, criar rotas de diferentes pontos de partidas, várias versões da simulação de tráfego do mesmo cenário e oferece ao usuário uma interface gráfica (KRAJZEWICZ, 2010), como mostra a Figura 1.

Na mesma, é possível verificar algumas das diversas opções de cenários que pode-se ter para realizar simulações, além dos objetos que compõe o mesmo, entre eles, veículos das mais variadas cores para algum tipo de estudo mais específico, vias de mão dupla e tripla, todas elas de cor preta, além cruzamentos e semáforos. A cor verde vista ao redor das vias, é a parte ignorada no cenário da simulação por não possuir vias para tráfego ou mesmo por tal região ser ignorada pelo usuário, quando o mesmo não quer determinado trecho no cenário de simulação.

Figura 1 – Ferramenta de simulação de tráfego SUMO.



Fonte: (SUMO DLR, 2019).

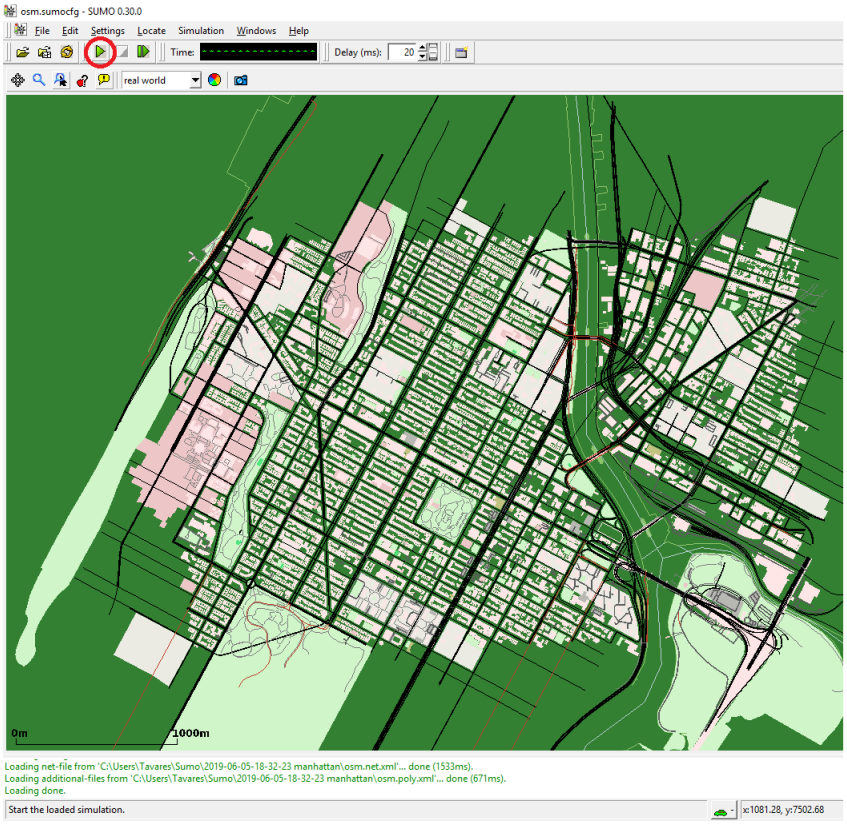
Com isso, foi realizada uma simulação de tráfego livre na ferramenta, possibilitando que sistemas de tráfego sejam modelados, incluindo diversos tipos de veículos, vias, semáforos, entre outras opções, permitindo assim, a obtenção de uma visão e análise do que realmente será executado pelo algoritmo, sendo assim possível analisar e verificar importantes estratégias para o planejamento de rotas (SUMO, 2018).

Para a simulação ser executada no SUMO, precisamos antes executar o *script* osmWebWizard que já vem disponível no arquivo de instalação da ferramenta para fazer a seleção de uma região, que por sua vez pode ser dimensionada a critério do próprio usuário como também a localização que deseja em qualquer parte do mundo, gerando arquivos contendo todas as informações com extensões net e cfg.

Após isso, o mapa é carregado com as estruturas e todas as características da região,

como quantidade de avenidas, semáforos, cruzamentos, entre outras, sem qualquer veículo. Nesse ponto, para que a simulação seja executada, precisa apenas que o usuário clique no botão Run, como pode ser visto na Figura 2. Após a saída do último veículo do cenário, então é encerrada a simulação gerando o arquivo log contendo informações dos eventos ocorridos como mostra a Figura 16.

Figura 2 – Carregamento do mapa e início da execução.



Fonte: (PRÓPRIO AUTOR, 2019).

Após o início da execução, é então feita a inserção de veículos no cenário a cada intervalo de tempo, de maneira contínua até que o último veículo consiga chegar ao seu local de destino. Todas as vias já são pré-configuradas ao se obter o mapa da região, com a velocidade máxima permitida, número de faixas e sentindo. Cada veículo tem sua velocidade de forma dinâmica, mas sempre respeitando o limite permitido por cada via.

O tempo em questão se dá em segundos, mas vale ressaltar que essa unidade de medida se trata de segundos em tempo de simulação. Neste caso, tem-se um índice chamado fator de tempo real que traz uma correlação entre o tempo de simulação e o tempo de processamento podendo ser definido no SUMO quanto vale 1 *step* de simulação e a partir da quantidade de *steps*, é possível calcular a quantidade de tempo de cada simulação.

O parâmetro *Step* utilizado é definido na inicialização do SUMO, que já está configurado com o valor 1, que equivale a 1 segundo, logo, ao se passar 3600 *Steps* é o mesmo que passar

3600 segundos também, porém convém ressaltar que não significa o tempo em que a simulação foi computada, porque neste caso o tempo é muito menor por se tratar de um universo diferente, criado apenas para fins de estudos.

2.2 XML

XML ou *Extensible Markup Language* é uma metalinguagem, que a W3C (*World Wide Web Consortium*) mantém desde o seu desenvolvimento em 1996, que descreve documentos XML do tipo classe de objetos de dados e determina as ações dos programas que os processam. Possui um formato de texto muito flexível e simples, desenvolvido para enfrentar os desafios das publicações eletrônicas e de suma importância na troca de uma imensa variedade de dados na Web (FERNANDES, 2006). Ainda permite acrescentar marcas ou rótulos do documento para a apresentação do conteúdo ou para dar semântica, que por muitas vezes é do tipo texto.

Fernandes (2006) detalha que o XML permite ao autor do documento total domínio na sua projeção de marcação. Isto é, os documentos são escritos na forma de linguagem de marcação, eles possuem metadados e dados, neste caso os metadados tem informações adicionais que dão um contexto ou sentido aos dados.

Assim, descreve dados estruturados que diferentes tipos de aplicativos podem manipulá-los, de forma a representar conteúdo especialmente na Web (SIAU; HINDE; STONE, 2012).

Lima e Carvalho (2005) descreve que, trata-se de um documento legível para os seres humanos e que pode ser também tratado por máquinas facilmente, devido ao conjunto de marcadores que é flexível e expressivo que o XML tem. Possibilitando assim, serem definidos dependendo da necessidade do usuário e do contexto do arquivo.

Na Figura 3, temos um exemplo de arquivo osm.net.xml gerado pelo SUMO, onde é possível fazer a extração dos dados necessários como as conexões das vias através da tag *from* e *to*, do tamanho da via através da tag *length*, velocidade com o tag *speed* e o nome da via com a tag *name*. Esses foram os atribuitos utilizados para a implementação do algoritmo e os mesmos são fornecidos para a realização de cálculos de maneira sequencial e paralela, podendo dessa maneira, ser tomada as melhores decisões de rotas possíveis..

Figura 3 – Arquivo .net.xml gerado na obtenção do mapa.

osm.net.xml

```
38496  <edge id="195743202" from="42437188" to="42437196" name="West 116th Street"
38497      priority="6" type="highway.tertiary">
38498      <lane id="195743202_0" index="0" disallow="tram rail_urban rail rail_electric ship"
38499          speed="22.22" length="121.97" shape="2120.11,4976.33 2011.75,5032.32">
38500          <param key="origId" value="195743202"/>
38501      </lane>
38502  </edge>
```

Fonte: (PRÓPRIO AUTOR, 2019).

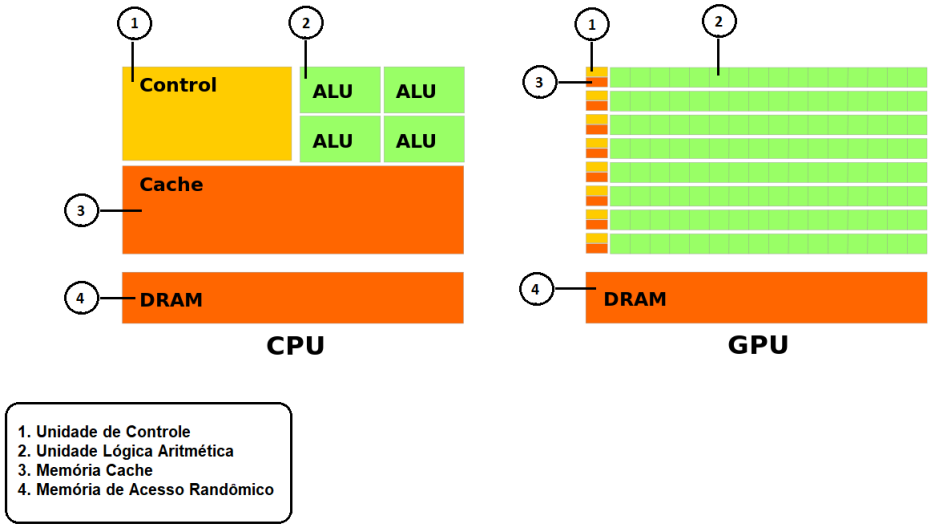
2.3 GPU

As aplicações gráficas tiveram dedicação por muitos anos dos processadores gráficos, mas com o passar do tempo a GPU teve grande avanço para o paralelismo, *multi-thread* e vários processadores devido a demanda por gráfico 3D de alta definição nos computadores, principalmente em ambiente de jogos.

Com relação a CPU tradicional, a GPU possui uma grande largura de banda de memória e alto poder computacional quando comparada a CPU (DETOMINI, 2010). Pois a programação paralela tem como finalidade reduzir o tempo total de execução de uma determinada aplicação por dividir suas tarefas (RAUBER; RÜNGER, 2013). A seguir na Figura 4, as arquiteturas da CPU e da GPU, respectivamente são apresentadas.

Observa-se que a arquitetura da CPU possui uma unidade de controle maior que na arquitetura da GPU, porém a quantidade de Unidade Lógica Aritmética da GPU, que por sua vez executa operações aritmética e lógicas é muito maior que da CPU. No caso da memória Cache, é utilizada pela CPU com o objetivo de reduzir a latência de acesso à memoria, devido ao seu tamanho, acaba ocupando boa parte do dispositivo, enquanto que na arquitetura da GPU a memória Cache tem seu tamanho muito reduzido, porém é compartilhada para aumentar a sua largura de banda, o que é relevante para cálculos com fluxos de dados muito grande. Ambas arquiteturas fazem uso da memória DRAM (Memória de Acesso Randômico), que é onde a informação precisa ser atualizada constantemente para que seja armazenada.

Figura 4 – Arquitetura CPU e GPU respectivamente.



Fonte: (WIKIMEDIA, 2019) Adaptada.

Um grande esforço se faz necessário na elaboração dos projetos para se obter uma otimização paralela na GPU, pois devem ser levados em consideração fatores como a sincro-

nização dos threads, distribuição eficiente do processamento de dados entre a CPU e a GPU, entre outros, para que problemas de otimização em larga escala em arquitetura GPU possam ser resolvidos de forma eficiente (LUONG; MELAB; TALBI, 2011). A NVIDIA, com o intuito de facilitar a programação na GPU, tornou ainda a arquitetura *Compute Unified Device Architecture* (CUDA) pública, que é baseada não apenas por algumas extensões como também na linguagem de programação C (NVIDIA, 2009).

Devido a essa evolução das GPUs, é possível que diversos aplicativos do mundo real sejam facilmente implementados e executados rapidamente em sistemas de vários núcleos. Os desenvolvedores de software, cientistas e pesquisadores estão descobrindo usos amplamente variados para a computação com GPU CUDA. Seu uso pode ser aplicado desde identificação de placas ocultas em artérias, análise do fluxo de tráfego aéreo, criptografia, bioinformática, entre outros (DETOMINI, 2010).

2.4 Grafos

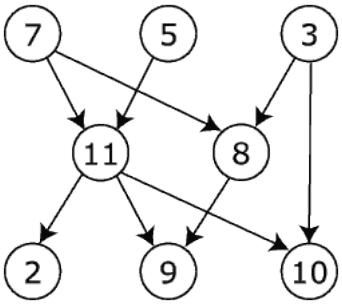
Com grafos, é possível fazer a modelagem como uma rede em que o tráfego de veículos tem um fluxo de origem e destino. Uma ferramenta matemática que possibilita uma caracterização de uma rede, dando a capacidade de estudar o desenvolvimento temporal, as propriedades topológicas, além de elementos relacionados ou não que fazem parte de um determinado contexto (SOUSA; ARAÚJO; MIRANDA, 2017).

Um grafo é denotado por G = (V,E) em que há dois conjuntos, onde um conjunto finito V de elementos chamados de vértices e um conjunto finito E de elementos chamados de arestas. Cada aresta pode ser identificada por possuir um par de vértices nas suas extremidades, se as arestas de um grafo G possuírem identificação com pares ordenados de vértices, então ele é dito ser um grafo orientado ou direcionado.

Do contrário, ele é dito um grafo não orientado ou não direcionado (THULASIRAMAN; SWAMY, 2011). Na Figura 5 tem-se um exemplo de um grafo orientado, ou seja, um dígrafo que é definido em Szwarcfiter (1988) como D(V,E) sendo um conjunto não vazio V de vertíces e um conjunto E de arestas, como pares ordenadas de vértices distintos. Dessa forma, em um dígrafo cada aresta(v,w) tem uma única direção partindo de v e chegando em w. A Figura 6 mostra um exemplo de como é um grafo não orientado, isto é, não possui nenhuma direção ou ordem a ser seguida.

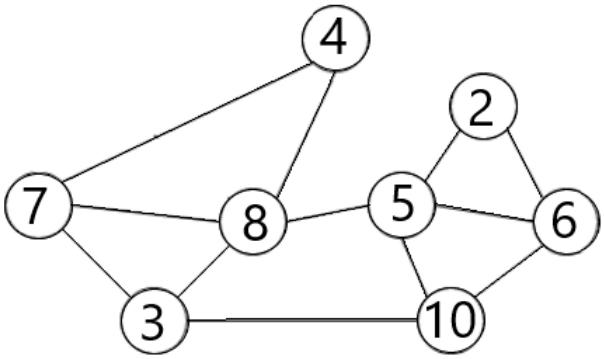
Nesse sentido, a modelagem da rede viária pode ser feita de maneira que as vias correspondem as arestas e suas intersecões, os cruzamentos representados como seus vértices respectivamente. Assim, o planejamento de rotas busca o caminho mínimo de acordo com os estados das arestas que possuem algum tipo de custo associado. Neste caso, a rota ótima é quando a soma dos seus custos de deslocamento é o menor, a partir da origem até o destino, avaliando todas as alternativas possíveis (FERGUSON; LIKHACHEV; STENTZ, 2005).

Figura 5 – Representação do grafo orientado.



Fonte: (WIKIPÉDIA, 2019).

Figura 6 – Representação do grafo não orientado.



Fonte: (PRÓPRIO AUTOR, 2019).

2.5 *A-Star*

Em Qian et al. (2018), é dito que o algoritmo *A-Star* (“A*” ou A Estrela) na literatura é indicado para lidar com soluções de rotas e, para encontrar um caminho de baixo custo, por ser um algoritmo heurístico eficiente muito utilizado para esse fim. Por explorar cada nó da configuração espacial e expandir o nó mais promissor em relação ao objetivo, ele é definido então como um algoritmo BFS (*Best-First Search*) (DUCHOŇ et al., 2014). Para que isso seja possível, Hart, Nilsson e Raphael (1968) afirma que uma função de avaliação é utilizada para então submetê-la a cada nó, para depois expandir o nó que retornar o menor custo na função.

Para direcionar e determinar a ordem na qual a busca explora os nós em um grafo que representa o cenário a ser avaliado, a função de avaliação é denotada por F(n). Ela é dada como F(n) = G(n) + H(n) em que o G(n) é o custo real do caminho ideal do ponto inicial para o ponto n. O custo estimado do caminho ótimo do ponto n para o ponto de destino é representado por H(n) na função, que por sua vez depende da informação heurística da área do problema.

O algoritmo *A-Star* pode ser vinculado a cada veículo do sistema de tráfego de uma cidade inteligente, gerando requisições de poder computacional muito alta, pois em busca da melhor solução é executado uma grande quantidade de cálculos. Os *clusters* e supercomputadores

podem ser utilizados nesse cenário por possuírem avançados recursos computacionais quando se trata de capacidade de processamento para se atender a tamanha exigência, pois pode-se utilizar processadores com alto desempenho para os cálculos desejados.

Em adicional a estrutura fornecida por ambos, pode-se fazer uso da programação paralela, onde tarefas podem ser executadas de forma independente por vários núcleos de processamento, além dos recursos tecnológicos de Computação de Alto Desempenho (HPC, *High Performance Computation*).

Contudo, *clusters* e supercomputadores são baseados em uma arquitetura de hardware com alto custo de implementação (BACELLAR, 2009), por utilizarem muitos computadores interligados operando juntos em uma tarefa. Neles, cada computador é chamado de nó (ou node). Os nós são conectados por conexões de altíssima velocidade, evitando dessa forma que o gargalo de conexão entre os nós prejudique o poder de processamento na troca de informações entre os mesmos.

A utilização de GPU's é uma alternativa viável de custo muito inferior se comparado a *clusters* e supercomputadores. São Hardwares específicos com placa de vídeo da NVIDIA que utiliza CUDA, uma extensão da linguagem de programação C para realizar as operações. Com isso, quando o algoritmo é bem desenvolvido e implementado, pode fazer uso da vantagem do processamento do HPC.

O algoritmo *A-Star* foi implementado de forma sequencial e paralelizada nesse trabalho, de forma que não apenas um único, mas sim vários veículos simultaneamente tenham suas rotas calculadas, na tentativa de atender as requisições usando o máximo possível de poder de processamento do hardware além do paralelismo, para que um desempenho satisfatório seja alcançado.

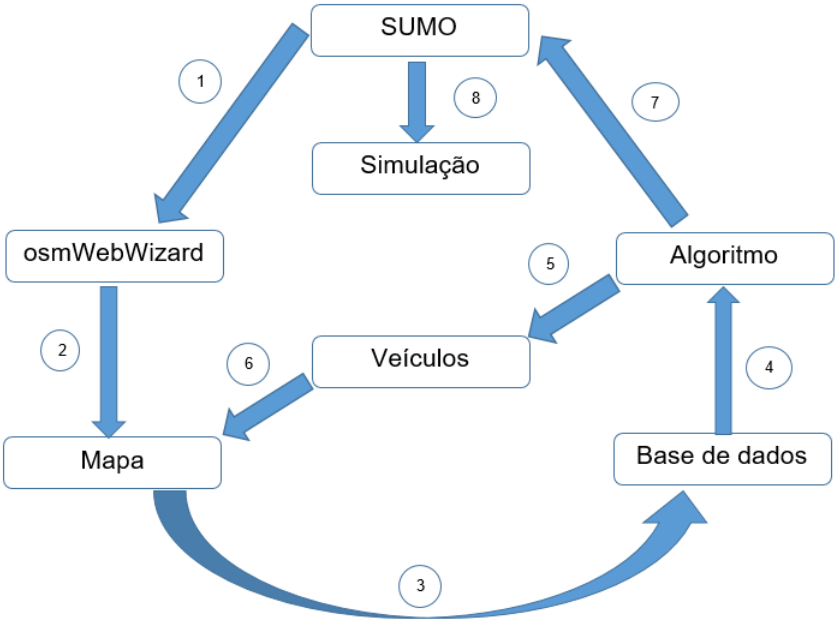
2.6 Diagrama de simulação no SUMO

Para um melhor entendimento do funcionamento da ferramenta de simulação de trágefo SUMO, é mostrado na Figura 7 um diagrama no qual contém também a ordem de execução de cada passo. Após a instalação do SUMO no computador, deve ser então executado o script osmWebWizard para fazer a captura da região desejada para realizar a simulação. Após a captura da região, é obtida a base de dados que contém todos os dados necessários, entre eles conexões das vias, faixas, entre outros, para serem inseridos no algoritmo.

No algoritmo é possível determinar a quantidade de veículos que participaram da simulação, como também os períodos em que os mesmos são inseridos no cenário. Após a realização do cálculos das rotas de cada veículo e também sua chegada ao destino requisitado, é feito a geração também de dados contendo todos os trajetos seguidos, abastecendo ainda a base de dados que será utilizada também pelo algoritmo. Logo após a obtenção de todos os dados necessários, é

realizada a chamada do algoritmo pela ferramenta de simulação ao se iniciar a execução da simulação.

Figura 7 – Diagrama de execução da simulação.



Fonte: (PRÓPRIO AUTOR, 2019).

3

Implementação e avaliação de algoritmos sequencial e paralelo

Esse capítulo descreve como se deu a análise e implementação do algoritmo sequencial e paralelo na busca da melhor rota, seguindo a metodologia descrita na seção 1.1.2 e sua aplicaçao na ferramenta de simulaçao de tráfego SUMO. Visando melhorar a qualidade de vida da população no que tange a mobilidade urbana, isto é, reduzindo o tempo de processamento nos cálculos necessários para oferecer rotas de menor custo nas cidades inteligentes, por dispor de uma infraestrutura necessária para a troca de informações entre veículos e centrais de processamento, auxiliando dessa forma o gerenciamento de sistemas de tráfego de determinada região.

3.1 Materiais e métodos

Para realizar o desenvolvimento e a implementação dos algoritmos como solução proposta neste trabalho, contou-se com a infraestrutura disponibilizada pelo ELAN (Experimental Laboratory in computer Networks) localizado no departamento de computação. Foi utilizado um computador da Asus com Sistema Operacional Microsoft Windows 10 64bits, processador Intel™ Core™ i5-7200U CPU @ 2.50GHz, com 8,0GB de memória RAM, com placa de vídeo Nvidia GeForce GTX 940MX, com 384 CUDA Cores e 2GB de memória, que permite paralelizar cálculos devido a sua GPU.

A implementação no computador citado logo acima, se deu para compreender toda a estruturação dos algoritmos e problemática envolvida, para logo após então implementá-los no *cluster* para poder realizar cálculos de maior escala devido ao seu maior poder computacional, uma vez que o mesmo possui uma arquitetura muito mais avançada.

As linguagens de programação utilizadas foram C, Python e CUDA da NVIDIA, com o editor de código-fonte gratuito Notepad++, Visual Studio Code, além dos compiladores GCC e G++. As linguagens C e Python foram escolhidas por serem mais próximas de CUDA, que como dito acima, pode acelerar a execução dos aplicativos de computação e também por

terem bibliotecas para facilitar a implementação de algumas funcionalidades. A linguagem C foi escolhida para fazer a implementação do algoritmo *A-Star* sequencial e CUDA para a implementação do algoritmo *A-Star* paralelizado.

A linguagem Python foi utilizada devido ao fato de já ser integrada ao SUMO e dispor de bibliotecas que facilitam o processo de extração dos dados necessários do arquivo osm.net.xml que é gerado automaticamente pelo SUMO. O arquivo osm.net.xml contém os atributos como velocidade do veículo, tamanho da via, pontos de conexão entre uma via e outra, além das permissões e também restrições, tanto no âmbito de deslocamento como posicionamento dos itens no mapa.

Logo após a validação ter sido concluída, foi então feita a simulação do tráfego de veículos com os cálculos de rotas ótimas, no *cluster* disponibilizado pelo LCAD (Laboratório de Computação de Alto Desempenho), que conta com uma arquitetura paralela composta por 27 nós de processamento com dois processadores Intel Xeon Ten-core E5-2660v2, 64 GB de memória RAM, disco SSD de 160 GB cada.

Desses, 5 nós de processamento cada um conta com 2 GPUs Nvidia Tesla K20. O *cluster* ainda possui um nó de storage com capacidade de armazenamento de 128TBytes, 128 GB de memória RAM, dois processadores Intel Xeon Six-core E5-2620, além de contar com o sistema operacional CentOS 7 de 64bits.

3.2 Modelagem da solução proposta

Para realizar o desenvolvimento do algoritmo *A-Star* em uma pequena região do bairro Siqueira Campos, como mostra a Figura 8 o espaço geográfico do mesmo, compreendeu-se que as ruas de mão dupla e mão única tornam o grafo em um dígrafo, isto é, um grafo direcionado devido o fato das vias possuirem sentidos.

O diagrama de ruas exposto na Figura 8 foi obtido a partir do Google Maps, como também os pontos de coordenadas latitude-longitude de cada cruzamento e as distâncias entre dois cruzamentos vizinhos. Tais valores foram inseridos na base de dados para então serem utilizados pelo algoritmo *A-Star* desenvolvido.

O *A-Star* é um algoritmo guloso de busca em grafos, que faz uso de heurísticas para auxiliar na ordem do processamento dos nós, com o objetivo de diminuir o tempo de processamento, indicado quando a busca pelo caminho mais curto não é ótimo, ou seja, rápido o suficiente (JASIKA et al., 2012). O mesmo, foi criado para tentar solucionar o problema de tempo de processamento de algoritmos que não faz uso de heurísticas para calcular a melhor rota, como o de Dijkstra (CRAUSER et al., 1998).

Figura 8 – Mapa do bairro Siqueira Campos, Aracaju - SE, Brasil.

Fonte: (GOOGLE MAPS, 2019).

O algoritmo de Dijsktra visita os vértices no grafo começando com o ponto de origem do objeto e em seguida, repetidamente, avalia o vértice mais próximo ainda não avaliado, colocando-os em um conjunto de vértices ainda a serem avaliados. Para atingir o ponto de destino, ele se expande para fora do ponto de origem, garantindo encontrar um caminho de menor custo para esse caminho se nenhuma aresta (via) tiver um custo negativo (DENG et al., 2012).

Após análises das principais características dos algoritmos *A-Star* e Dijkstra, devido suas complexidades, optou-se pela utilização do algoritmo *A-Star*, o qual foi aplicado a problemática do bairro Siqueira Campos ilustrado na Figura 8 como seu validador. Através do cálculo da distância média a partir das coordenadas de latitude e longitude, essa métrica foi então aplicada no algoritmo para calcular as possíveis melhores rotas nessa região, no menor tempo possível.

As coordenadas e as distâncias médias entre cruzamentos das vias, foram obtidas através do Google Maps. Para que seja possível o tráfego em apenas um sentido, os pontos geográficos dos cruzamentos das vias podem ser mapeados em vértices de um grafo e as vias em arestas. No desenvolvimento do algoritmo foram utilizadas as coordenadas dos vértices como par cartesiano XY, sendo necessário transformar as coordenadas em unidades de latitude e longitude para pares ordenados XY.

Inicialmente obtêm-se os dados em latitude e longitude, para só então fazer um mapeamento desses pontos em um sistema de coordenadas esféricas. Dessa forma, a coordenada latitude será mapeada no ângulo θ, da forma transladada θ = 90 - Latitude e para longitude será mapeada em ϕ da forma ϕ = Longitude. A terra teve seu raio considerado como r = 6.373km, permitindo mapear os vértices em um eixo XYZ utilizando a transformação de coordenadas.

$$\begin{cases} x = rsen\theta cos\phi \\ y = rsen\theta sen\phi \\ z = rcos\theta \end{cases} \quad (1)$$

Após a transformação, é necessario que os eixos sejam rotacionados de maneira que o eixo Z aponte para algum vértice. Assim, todos os vértices tem praticamente o mesmo valor de Z e as suas coordenadas podem ser mapeadas utilizando os pares ordenados XY. Então para descobrir a melhor rota a ser seguida entre uma origem e destino, o algoritmo realiza cálculos da heurísticas entre vizinhos para que os movimentos tenham seus custos sabidos.

Com a soma entre o custo para chegar no próximo vértice e o custo da heurística entre esse próximo vértice e o ponto de destino obtêm-se então o menor custo. No momento que o sentido de uma via proibida é indicado devido ao custo entre vértices vizinhos ser menor, é atribuído então um valor muito alto de maneira que o custo para seguir nessa via proibida seja inviável, evitando dessa forma a utilização de técnicas de backtracking por ser muito custosa para o processamento.

A distância euclidiana foi utilizada como base para o cálculo da heurística, pois pode ser calculada facilmente subtraindo o vetor que aponta para o ponto de origem e o vetor que aponta para o ponto de destino, supondo assim, o cálculo em um sistema geométrico euclidiano aproximando a região da terra a um plano.

Tal aproximação é válida quando se trata da problemática do tráfego nas cidades, por terem suas distâncias pequenas quando comparadas ao raio da terra. Assim, quando se faz o cálculo da distância a curvatura da terra não exerce influência de forma significativa, porém em roteamento de aviões, sua influência tem impactos na distância.

Além disso, o fato de que cada vértice do grafo pode ser mapeado em um plano cartesiano é outra vantagem em adotar o método da distância euclidiana, pois assim, a partir de um par cartesiano XY cada ponto pode ser identificado e ainda as informações dos pares ordenados XY dos seus respectivos pontos podem ser compartilhadas entre os veículos envolvidos no problema, permitindo dessa forma a troca mútua de dados entre várias instâncias de processamento independente.

Com a definição da forma de cálculo da heurística, pode-se então efetuar a criação do algoritmo do tipo *A-Star* para calcular a melhor rota em um grafo. O algoritmo parte de um vértice inicial do grafo e calcula o custo para alcançar um vértice vizinho F(n) que esteja conectado a si por uma aresta.

A soma entre o custo para alcançar um novo vértice G(n) mais o custo da heurística desse vértice H(n), equacionados na forma F(n) = G(n) + H(n), fornece dessa forma o custo. O algoritmo *A-Star* é indicado nestes casos por fazer buscas de um caminho em grafos, de um ponto de origem até um ponto de destino, fazendo uso de heurísticas fornecidas pelo cenário.

3.3 Algoritmo *A-Star* sequencial

O pseudocódigo para o cálculo da heurística baseado em distância euclidiana se encontra no Algoritmo da função Calcular, como mostra a Figura 9.

Figura 9 – Algoritmo da função Calcular

```
Entradas: vH: Matriz contendo os pares ordenados XY para cada
              ponto do grafo, as linhas representam o vértices
              do grafo e a coluna 1 para x e 2 para y.
          fim: O vértice de destino do veículo.
            i: O vértice que se quer calcular a Heurística.
        carro: O veículo que se quer calcular a rota.
        CH: A matriz que armazena o resultado da Heurística.
1:  x = vH[ fim ][ 1 ] – vH[ i ][ 1 ]
2:  y = vH[ fim ][ 2 ] – vH[ i ][ 2 ]
3:  CH[ carro ][ i ] = sqrt( x*x + y*y )
```

Fonte: (PRÓPRIO AUTOR, 2019).

Após calcular os novos vértices, o algoritmo deve buscar aquele com o menor custo e então repetir os cálculos considerando esse novo vértice como parâmetro. O código segue de vértice em vértice e encontra uma concisão de parada quando o próximo vértice é o destino da rota. O pseudocódigo desse algoritmo se encontra exibido no Algoritmo *A-Star* sequencial e no paralelizado, pois a função Calcular é utilizada por ambos.

O Algoritmo *A-Star* sequencial tem como dados de entrada principais a matriz de vizinhança e a matriz que armazena os pares XY. A matriz que contém os pares XY tem um número de linhas igual a quantidade de vértices do problema e possui duas colunas, uma para o ponto cartesiano x e outra para o y. Essa matriz permite o cálculo da heurística a partir da distância euclidiana.

Figura 10 – Algoritmo *A-Star* sequencial

Entradas: **vH**: Matriz contendo os pares ordenados XY para cada ponto do grafo, as linhas representam os vértices do grafo e a coluna 1 para x e 2 para y.
G: Matriz vizinhança que contém custo para sair de um vértice *i* e chegar a um vértice *j*.

```
1: PARA (k = 1 : Nº carros )
2:       inicio = IniFim[k][0]
3:       fim = IniFim[k][1]
4:       pc[inicio][k]=1
5:       ponto=inicio
6:       Enquanto (ponto ≠ fim)
7:          PARA (i = 1: Nº Vértices)
8:             SE ( G[ponto][i] ≠ 0.0  e  pc[i][k] ≠  1)
9:                Calcular (x , y)
10:               SE ( CG[i][k] > (G[ponto][i]+CG[ponto][k]) ou vcusto[i][k]=0.0)
11:                  CH[i][k] = sqrt(x*x+y*y)
12:                  CG[i][k] = G[ponto][i]+CG[ponto][k]
13:                  vcusto[i][k]=CG[i][k] + CH[i][k]
14:                  P[i][k]=ponto
15:               FIM – SE
17:            FIM – SE
18:         FIM – PARA
19:         PARA (i = 1: Nº Vértices)
20:            SE ( pc[i][k] ≠ 1 , vcusto[i][k]  e vcusto[i][k]≠0.0)
30:               menor = vcusto[i][k];
31:               imenor = i;
32:            FIM – SE
33:         FIM – PARA
34:         ponto = imenor
35:         pc[ponto][k]=1;
36:      FIM – ENQUANTO
36: FIM – PARA
```

Fonte: (PRÓPRIO AUTOR, 2019).

A matriz vizinhança é uma matriz quadrada com dimensão igual ao número de vértices do grafo. Cada posição (i , j) contém o custo para sair do vértice i e chegar ao vértice j que são vizinhos entre si. Caso os vértices não sejam vizinhos é atribuído um valor 0 que significa que é impossível sair de i e alcançar j diretamente. Essa notação exige a utilização do mecanismo de backtracking caso exista contramão da via. Dessa forma, o grafo se torna um dígrafo e sempre que encontrar um beco sem saída deve retornar a um ponto livre.

Como já mencionado, o mecanismo de backtracking soluciona o problema da contramão, entretanto esse tipo de algoritmo tem um alto custo em complexidade e poder computacional, dessa forma é possível transformar o dígrafo novamente em um grafo, atribuindo um custo excessivo a uma tentativa de seguir em contramão. Na prática, um caminho que inclua um trecho em contramão terá um custo muito alto se comparado com outros caminhos mais longos, porém

sem contramão.

O Algoritmo *A-Star* sequencial permite sair de um ponto de origem para um ponto de destino e calcula o custo, usualmente o caminho percorrido deve ser obtido calculando-se o custo de deslocamento partindo do vértice de destino até ao vértice de origem. Esse método de obtenção do caminho utiliza algoritmos backtracking da mesma forma que ocorre quando se alcança uma via de contramão. Com isso, para evitar o problema de backtracking, é possível criar um vetor com a mesma quantidade de vértices. Vale ressaltar que em caso de encontrar dois caminhos ótimos de custos iguais, o último encontrado que é válido.

As posições do vetor representam os vértices e o conteúdos armazenados nessa posiçoes representam os vértices de onde o veículo deve vir para alcançá-lo. Assim, o vetor com descrição P[4] = 8, por exemplo, significa que o veículo partiu do vértice 8 e chegou ao vértice 4. Deste modo, um caminho com origem no vértice 2 e destino no vértice 4 e que tem como caminho 2-1-7-6-8-4 teria a seguinte sequência de valores em seu vetor posição: P[1] = 2, P[7] = 1, P[6] = 7, P[8] = 6 e P[4] = 8. A representação vetorial desse caminho é demonstrada na Figura 11.

Figura 11 – Vetor que armazena o melhor caminho encontrado

Posição do i vetor	Conteúdo da posição i
1	2
2	-
3	-
4	8
5	-
6	7
7	1
8	6
9	-

Fonte: (PRÓPRIO AUTOR, 2019).

3.4 Algoritmo *A-Star* paralelizado

Esse problema foi também abordado na perspectiva de processamento paralelo. Dessa forma, foi desenvolvido um algoritmo utilizando a linguagem CUDA apresentado na Figura 12

para dividir as instâncias de processamento entre os vários *threads* da placa de vídeo. Assim em uma busca em grafos, como no caso de roteamento de tráfego, as instruções do algoritmo *A-Star* são as mesmas, independente do veículo, alterando-se apenas os dados de entrada e saída.

Figura 12 – Algoritmo *A-Star* paralelo.

Entradas: **H**: Matriz contendo os pares ordenados XY para cada ponto do grafo, as linhas representam os vértices do grafo e a coluna 1 para x e 2 para y.
G: Matriz vizinhança que contem custo para sair de um vértice *i* e chagar a um vértice *j*.
Calcular: é uma função que recebe os pontos X e Y e realiza a operação vetorial de medição de distância entre dois pontos.

```
1:  k = blockIdx.x
2:  inicio = IniFim[0+k*2]
3:  fim = IniFim[1+k*2]
4:  pc[k +inicio*( Nº carros)]=1
5:  ponto=inicio
6:  Enquanto (ponto ≠ fim)
7:      PARA (i = 1: Nº Vértices)
8:          SE ( G[i+ponto*( Nº Vértices)] ≠ 0.0  e  pc[k +i*( Nº carros)] ≠  1)
9:              Calcular (x , y)
10:             SE ( CG[k +i*( Nº carros)] > (G[i+ponto*( Nº Vértices)]+CG[k +ponto*( Nº carros)]))
11:                 CH[k +i*( Nº carros)] = Raiz(x*x+y*y)
12:                 CG[k +i*( Nº carros)] = G[k +i*( Nº carros)]+CG[k +ponto*( Nº carros)]
13:                 vcusto[k +i*( Nº carros)]=CG[k +i*( Nº carros)] + CH[k +i*( Nº carros)]
14:                 P[k +i*( Nº carros)]=ponto
15:             FIM – SE
17:         FIM – SE
18:     FIM – PARA
19:     PARA (i = 1: Nº Vértices)
20:         SE ( pc[k +i*( Nº carros)] ≠ 1 , vcusto[k +i*( Nº carros)]  e vcusto[k +i*( Nº carros)]≠0.0)
30:             menor = vcusto[k +i*( Nº carros)];
31:             imenor = i;
32:         FIM – SE
33:     FIM – PARA
34:     ponto = imenor
35:     pc[k +ponto*( Nº carros)]=1;
36: FIM – ENQUANTO
```

Fonte: (PRÓPRIO AUTOR, 2019).

Além disso, a informação sobre a estrutura do grafo é a mesma para todos os veículos e pode ser compartilhada entre eles. Dessa forma, o algoritmo paralelo desenvolvido dividiu o processamento entre os diversos veículos. A divisão dos *threads* na placa de vídeo se deu no formato em que um bloco trata do problema de roteamento de um veículo e cada bloco continha um *thread* de processamento.

Neste caso, a separação do cálculo de cada rota para cada veículo em *threads* de processamento separadas, a paralelização se torna mais simples e também eficiente. Assim, o algoritmo paralelo fica muito parecido com o sequêncial, modificando somente a variável k que está

representando os veículos.

Para fazer essa alteração na linguagem CUDA, basta mudar o indexador k dos veículos pelo identificador do *thread* de processamento. Dessa forma, apenas cada veículo tem seu cálculo realizado pelas *threads* de cada bloco de processamento, bastando colocar blockIdx.x (indexação CUDA) no lugar de k como descrito na linha 1 na Figura 12. No momento do processamento, o cálculo da rota de um veículo tem dedicação de cada bloco da GPU e de forma simultânea todos os blocos fazem o mesmo.

3.5 Validação em módulo teste

O algoritmo *A-Star* foi inicialmente testado utilizando uma área do bairro Siqueira Campos, na cidade de Aracaju, estado de Sergipe devido ao fato do conhecimento ocular da região e também por contar com estruturas viárias semelhantes a de grandes centros, contando com semáforos, vias de mão dupla, quarteirões retangulares, entre outras. Essa região é composta por 33 vértices, ou cruzamentos, enumerados de 0 a 32. O deslocamento entre os vértices, ou deslocamento ponto-a-ponto, pode-se dar por vias de mão dupla e mão única.

As vias de mão única estão representadas por linhas verdes e seu sentido é indicado por setas, também verdes. As vias de mão dupla são representadas por linhas vermelhas. A região escolhida assim como o dígrafo e suas características estão apresentadas na Figura 13.

Figura 13 – Dígrafo representativo do bairro Siqueira Campos, Aracaju - SE, Brasil

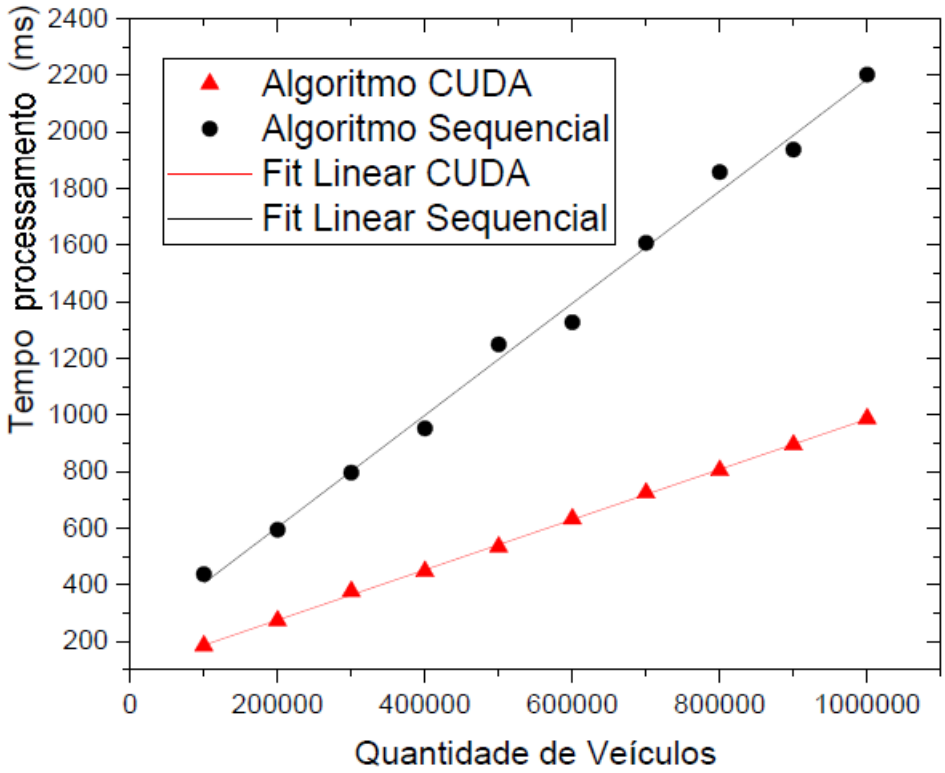
Fonte: (ADAPTADO GOOGLE MAPS, 2019).

O algoritmo desenvolvido foi aplicado ao dígrafo do bairro Siqueira Campos e apresentou resultados coerentes para rotas aleatórias conseguindo encontrar os caminhos corretos sem passar por vias impróprias. Em todos os casos analisados o caminho encontrado foi o caminho ótimo. Isso se deve ao fato de que a estrutura do bairro em questão é composta por quarteirões retangulares e com geometria não exótica.

Apesar da alta precisão no cálculo do caminho ótimo, não se pode afirmar que o algoritmo *A-Star* sempre encontra o caminho ótimo. Algoritmos como Dijkstra são conhecidos por conseguir esse feito, entretanto, na maioria dos casos, o algoritmo *A-Star* consegue encontrar o caminho ótimo e em menor tempo.

Foi feita uma comparação entre o poder de processamento do algoritmo sequencial e paralelizado, como pode ser observado na Figura 14. Efetuou-se o cálculo do roteamento de vários veículos onde se calculou o tempo de processamento. A quantidade de carros escolhidas foi extrapolada para que se pudesse obter um valor de tempo mensurável, visto que o baixo número de vértices e a boa otimização do algoritmo tornou o processamento bastante rápido.

Figura 14 – Gráfico do tempo em milissegundos em função da quantidade de veículos.



Fonte: (PRÓPRIO AUTOR, 2019).

A quantidade de veículos variou de 100 mil a 1 milhão, seguindo rotas baseadas em origens e destinos aleatórios para cada veículo, entretando os algoritmos implementados foram executados computando as mesmas rotas de cada veículo para efeito comparativo. O resultado

obtido está apresentado na Figura 14.

Os círculos pretos representam os pontos relacionados ao processamento sequencial realizados pelo processador da CPU, enquanto que os triângulos vermelhos representam pontos relacionados ao processamento paralelizado ocorrido na GPU. Foi observado que o processamento paralelo apresentou um comportamento linear, enquanto que no sequencial houveram oscilações, provavelmente devido a programas do próprio sistema operacional que executavam em plano de fundo.

O comportamento obtido demonstrou que os processadores da placa de vídeo estavam em pleno uso, e com pouca ou nenhuma ociosidade. Para realizar uma comparação quantitativa dos resultados foram feitos fits lineares em ambos os casos, sequencial e paralelo, onde foi obtido o coeficiente linear de ambas as retas. A relação entre esses coeficientes indica a proporcionalidade entre o tempo de execução das duas abordagens.

O fit linear é observado nas linhas cheias da Figura 14, sendo a vermelha com relação ao processamento paralelo (CUDA) e a linha preta o processamento sequencial. O coeficiente angular do processamento sequencial foi de 0,00198 enquanto que no caso paralelizado foi de 0,00089. Assim, o tempo de processamento sequencial é por volta de 2,2 vezes mais demorado que o tempo de processamento utilizando o algoritmo CUDA.

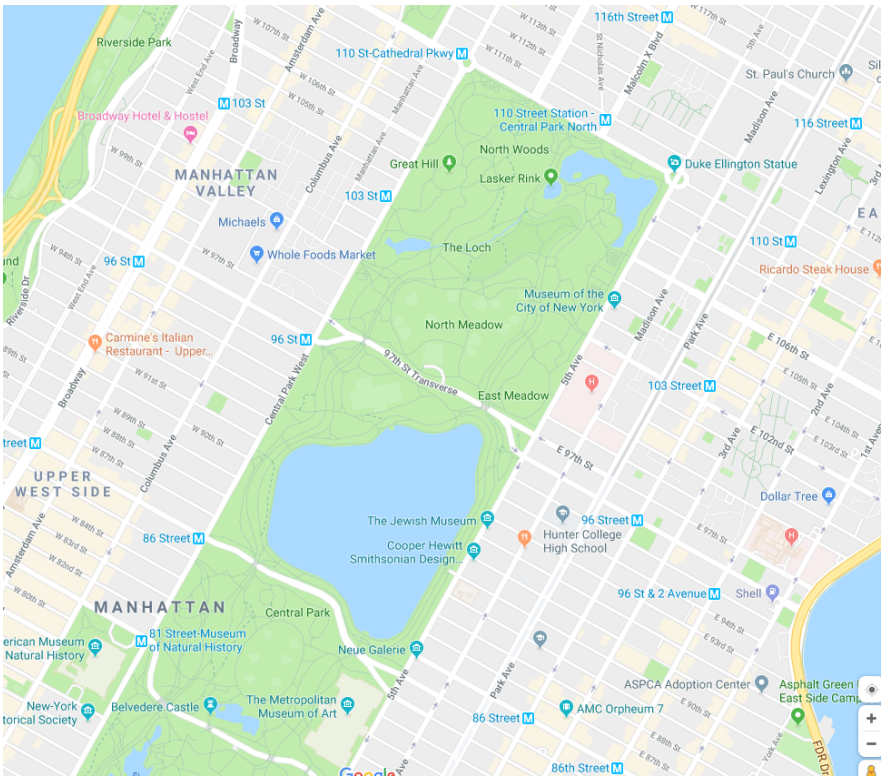
3.6 Simulação em Manhattan com aplicação em tempo real

Após a realização da validação no cenário de teste, que foi em uma região do bairro Siqueira Campos, na cidade Aracaju - SE e da depuração do código a fim de evitar *bugs*(erros), foi então realizada a simulação dos cálculos de rotas na ferramenta SUMO, em uma região de Manhattan, Nova Iorque - EUA, mais especificamente nas seguintes coordenadas de latitude 40.78343 e longitude -73.96625, como mostra a Figura 15.

A cidade de Manhattan tem vias caracterizadas por quarteirões retangulares na maior parte de sua extensão, esse tipo de estrutura torna mais fácil o estudo do comportamento de veículos em simulações, visto que esse comportamento se torna previsível. Estudos de casos inclusos nas bibliotecas do SUMO contêm ruas fictícias obedecendo as mesmas estruturas retangulares.

O caso real da cidade de Manhattan ainda inclui pequenas regiões que fogem das estruturas retangulares, o que é bem vindo pois torna possível observar a interação de regiões retangulares a partir de vias que não obedecem esse tipo de estrutura, possibilitando a observância dos eventos ocorridos, tanto quando ocorre de maneira esperada quanto uma situação de anomalia.

Figura 15 – Mapa de Manhattan, Nova Iorque - EUA.



Fonte: (ADAPTADO GOOGLE MAPS, 2019).

As simulações realizadas no SUMO, aconteceram de três maneiras diferentes, uma simulação com o algorimto *A-Star* sequencial, outra com o algoritmo *A-Star* paralelizado e outra de forma sequencial intrínseco da própria ferramenta, sem o algoritmo *A-Star* de modo que fosse possível fazer a coleta dos dados nos arquivos gerados, possibilitando fazer as comparações necessárias para as análises de viabilidade de qual algoritmo tinha melhor desempenho em cenários alternativos.

Os resultados mensurados desse trabalho são provenientes das simulações que foram executadas com intervalos de inserções dos véiculos de 0.4 até 2.0 segundos no cenário e com tempo de recálculo das rotas a cada 4 minutos. Dessa maneira, foi possível observar e avaliar o comportamento das execuções do algoritmo da própria ferramenta, o *A-Star* sequencial e o *A-Star* paralelizado.

A Tabela 1 nos mostra os períodos de tempo que foram utilizados juntamente com as suas respectivas quantidades totais de veículos, isto é, para cada período de tempo de inserção tinha-se o seu respectivo total de veículos utilizados por cada forma de execução dos algoritmos nas simulações.

Tabela 1 – Periodização e quantidade de veículos inseridos

Período	**Total**	**Período**	**Total**
2.0	1662	1.1	2997
1.9	1752	1.0	3297
1.8	1851	0.9	3668
1.7	1963	0.8	4135
1.6	2085	0.7	4739
1.5	2222	0.6	5542
1.4	2375	0.5	6655
1.3	2553	0.4	8275
1.2	2762		

Fonte: PRÓPRIO AUTOR.

Com o término das execuções das simulações, foram então coletados dados como tamanho médio das rotas (*RouteLength*), duração média das viagens (*Duration*), tempo médio de processamento e tempo total de processamento dos cálculos das rotas. Essas informações foram extraídas dos arquivos de log, todas elas expostas na Figura 16, gerados após a conclusão de cada simulação.

Figura 16 – Arquivo log gerado na simulação.

```
Performance:
 Duration: 1032855ms
 Real time factor: 4.8061
 UPS: 997.673439
Vehicles:
 Inserted: 1662
 Running: 0
 Waiting: 0
Teleports: 5 (Collisions: 4, Yield: 1)
Emergency Stops: 14
Statistics (avg):
 RouteLength: 4564.46
 Duration: 620.01
 WaitingTime: 279.36
 TimeLoss: 399.10
 DepartDelay: 1.62

AStarRouter answered 1 queries and explored 1372.00 edges on average.
AStarRouter spent 3ms answering queries (3.00ms on average).


Tempo Total Processando rotas:  0.018807172775268555
Tempo Medio Processando rotas:  0.0009403586387634278
```

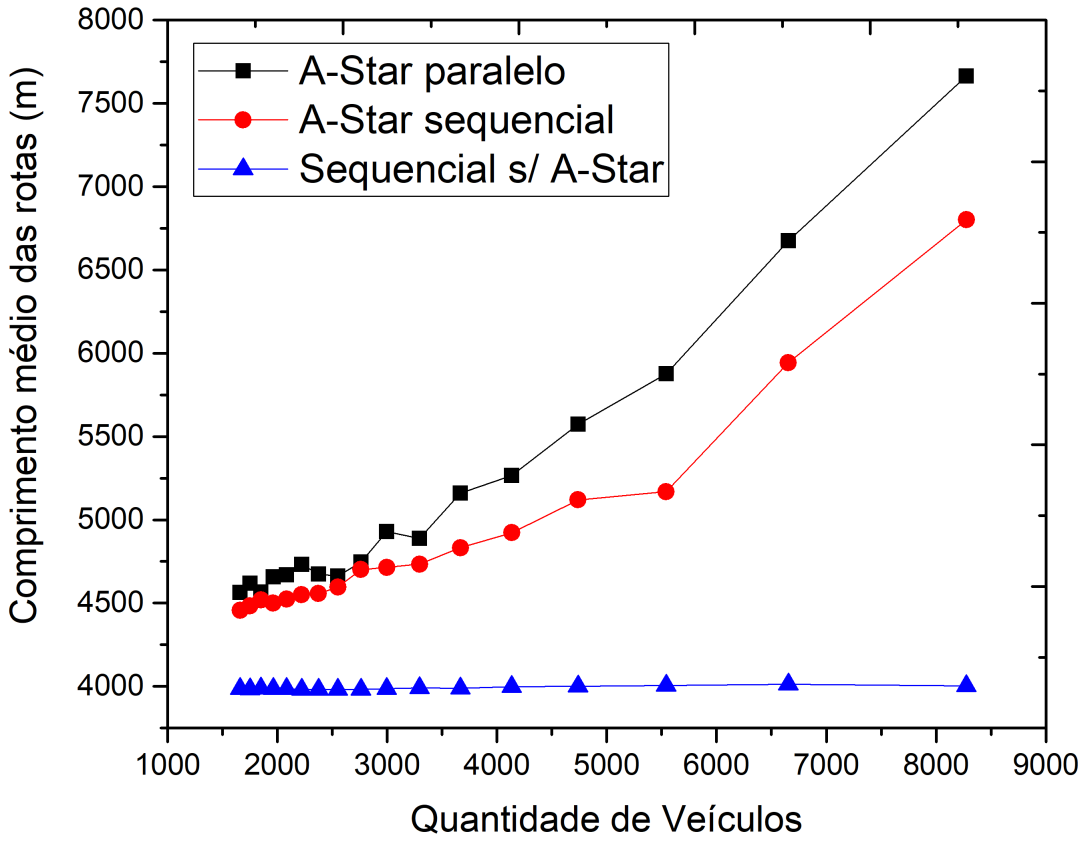
Fonte: (PRÓPRIO AUTOR, 2019).

De posse desses dados, foram realizadas comparações de todas as formas de execuções citadas acima, com o intuito de analisar qual melhor algoritmo se adequaria melhor em diversos cenários, devido a aleatoriedade que pode haver em qualquer um dos mesmos, como uma quantidade muito grande de veículos em uma região de um bairro em dado momento do dia, acidentes ocorridos em algumas vias causando congestionamentos e avenidas muito íngrimes.

3.6.1 Análise do comprimento médio das rotas

A primeira comparação realizada foi com relação ao tamanho médio em metros das rotas calculadas, que significam os tamanhos dos percursos traçados para os veículos. Como mostra a Figura 17, é possível notar o quão vantajoso foi o algoritmo sequencial da própria ferramenta de simulação, neste caso Dijkstra, quando comparado com o algoritmo sequencial com o *A-Star* e também com o *A-Star* paralelizado.

Figura 17 – Gráfico do comprimento médio das rotas em metros em função da quantidade de veículos.



Fonte: (PRÓPRIO AUTOR, 2019).

Inicialmente, quando executadas as simulações com um número consideravelmente pequeno de carros, 1662 veículos, o algoritmo sequencial da própria ferramenta já faz um cálculo do tamanho médio das rotas menor, enquanto o algoritmo *A-Star* sequencial tem desempenho quase similar ao do algoritmo *A-Star* paralelizado, com rotas maiores.

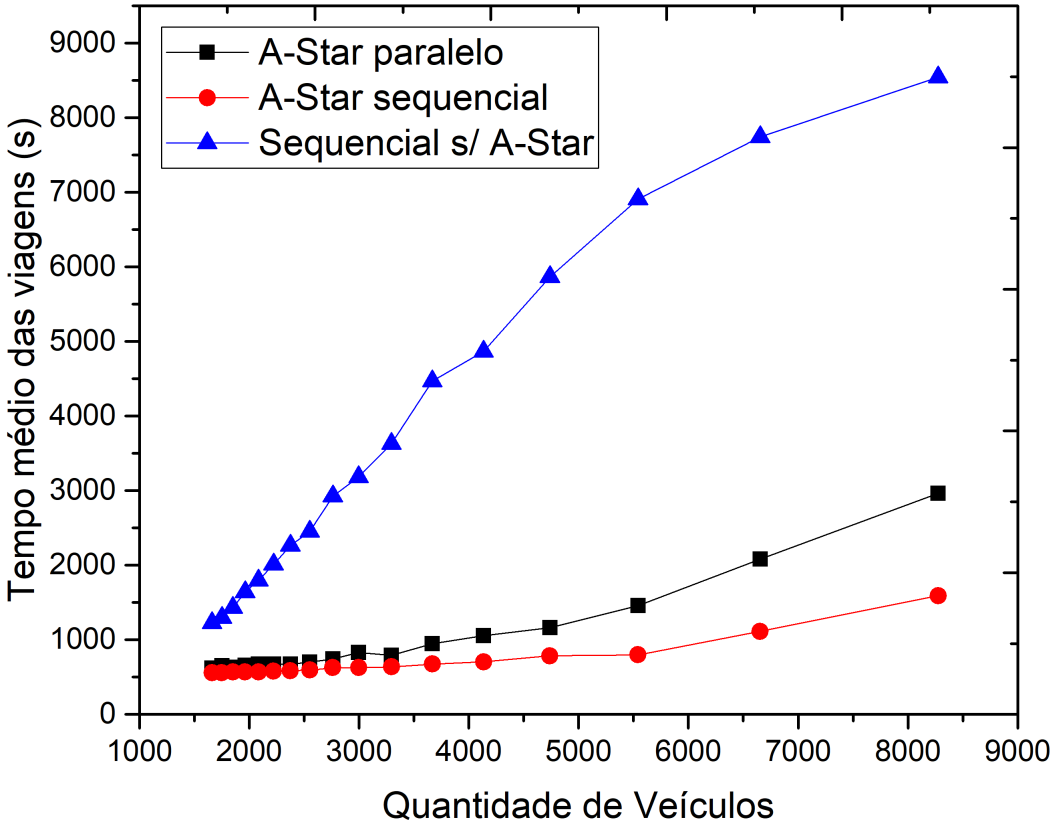
Contudo, ao chegar no final das execuções, onde a quantidade de veículos atinge a marca de 8275, o desempenho do algoritmo sequencial da ferramenta chega a ser bem próxmo de duas vezes melhor que o algoritmo *A-Star* paralelizado e uma vez e meia melhor com relação ao algoritmo *A-Star* sequencial. Esse desempenho abaixo do esperado nessa métrica dos algoritmos propostos por este trabalho, é creditado a heurística utilizada pelos mesmos em relação ao da

ferramenta SUMO.

3.6.2 Análise do tempo médio das viagens

Com os cálculos feitos e análise comparativa dos tamanhos médios das rotas, é a vez agora de comparar o tempo médio das viagens nas execuções das simulações contendo os três tipos de algoritmos. Já que se tem os tamanhos médios das rotas a serem seguidas, agora de fato, precisa-se analisar os tempos médios das viagens oriundos das respectivas rotas escolhidas. Essas durações médias das viagens se dão em segundos, como mostra na Figura 18.

Figura 18 – Gráfico do tempo médio em segundos das viagens em função da quantidade de veículos.



Fonte: (PRÓPRIO AUTOR, 2019).

Ao analisar o gráfico da Figura 18, percebe-se a vantagem obtida com o algoritmo *A-Star* sequencial em relação aos demais. Logo no início, com a quantidade de 1662 veículos, o algoritmo *A-Star* sequencial tem quase que o mesmo desempenho quando comparado com o algoritmo *A-Star* paralelizado, já com relação ao algoritmo sequencial sem o *A-Star*, o algoritmo *A-Star* sequencial tem praticamente o dobro de vantagem na duração da viagem.

Ao fim, quando atingida a quantidade de 8275 veículos, o algoritmo *A-Star* sequencial consegue uma vantagem ainda maior, em relação ao algoritmo *A-Star* paralelizado e também ao

algoritmo sequencial da própria ferramenta SUMO, alcançando uma diferença na vantagem de 7 vezes.

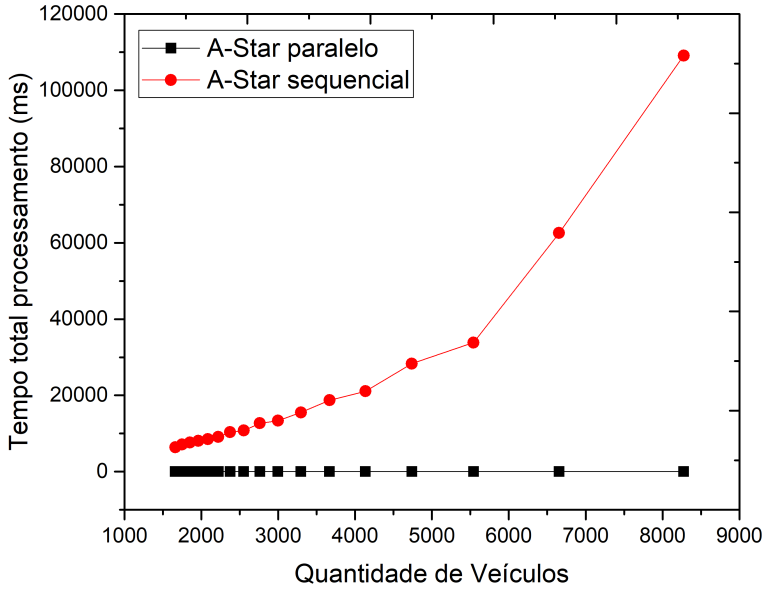
Com isso, temos que o algoritmo sequencial sem *A-Star*, tem melhor comportamento quando se trata de cálculos do tamanho médio das rotas, porém o mesmo tem uma grande desvantagem em relação aos demais algoritmos quando se trata da duração da viagem. Nesse quesito, o que teve melhor desempenho foi o algoritmo *A-Star* sequencial, como pode ser notado na Figura 18.

O algoritmo *A-Star* paralelizado neste quesito também teve um comportamento inesperado em relação ao sequencial, devido ao fato de fazer os cálculos das melhores rotas para todos os veículos de forma instantanea e retornar para os veículos no mesmo tempo, causando assim um congestionamentos em alguns momentos, gerando atrasos na viagem.

3.6.3 Análise do tempo total de processamento

A próxima métrica a ser analisada é o tempo total de processamento para os cálculos dos tamanhos médios das rotas e das médias das durações das viagens, para saber quanto tempo levam para os cálculos serem processados. Aqui, foram avaliados apenas os dois algoritmos, o *A-Star* sequencial e o *A-Star* paralelizado, devido ao fato do algoritmo sequencial da própria ferramenta de simulação não realizar recálculos das rotas, uma vez que ela já foi definida.

Figura 19 – Gráfico do tempo total em milissegundos de processamento em função da quantidade de veículos.



Fonte: (PRÓPRIO AUTOR, 2019).

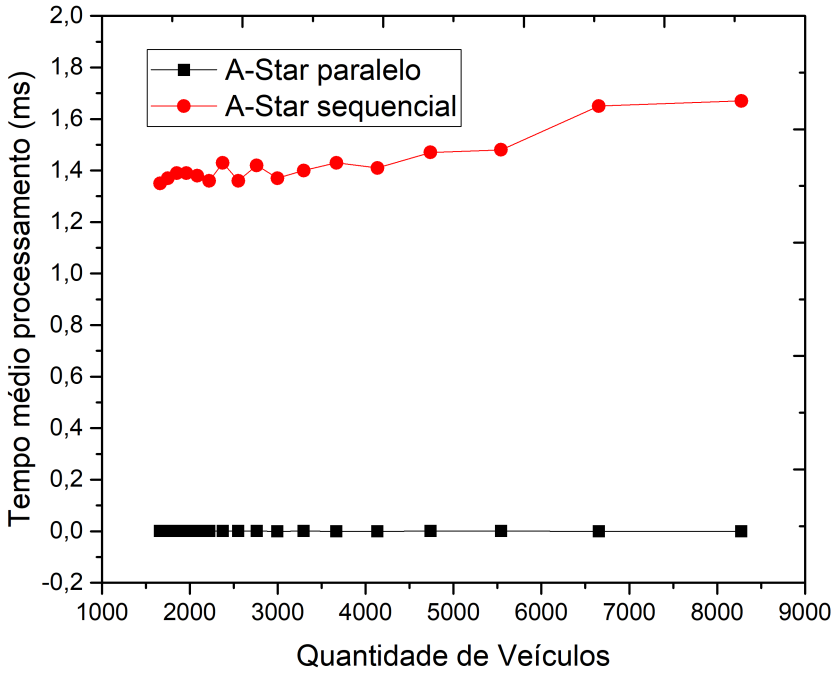
Assim, observando o gráfico da Figura 19 nota-se que o algoritmo *A-Star* paralelizado tem todas suas execuções mais rápidas do que o algoritmo *A-Star* sequencial. No início, quando

a quantidade de veículos é de 1662, a vantagem é de 321.000 vezes, quando a quantidade de veículos chega a 4739, a vantagem aumenta consideravelmente para 2.360.500 vezes mais rápido. Por fim, quando a quantidade de veículos chega na marca de 8275, a vantagem chega a 21.820.600 vezes mais rápido que o algoritmo *A-Star* sequencial, uma diferença abismal.

3.6.4 Análise do tempo médio de processamento

De posse do tempo total de cada execução, foi analisado também a métrica de tempo médio de duração do processamento para cada cálculo realizado pelos algoritmos, o *A-Star* sequencial e o *A-Star* paralelizado pelo mesmo motivo mencionado acima com relação ao algoritmo da própria ferramenta de simulação.

Figura 20 – Gráfico do tempo médio em milissegundos de processamento em função da quantidade de veículos.



Fonte: (PRÓPRIO AUTOR, 2019).

Aqui, como mostra o gráfico da Figura 20, a vantagem das execuções do algoritmo *A-Star* parelizado para o *A-Star* sequencial já é muito grande desde o início, quando a quantidade de veículos é de 1662. Nesta quantia a vantagem chega a ser de um 1350 vezes mais rápido. Quando a quantidade de veículos chega em 8275, a vantagem da execução do algorimto *A-Star* paralelizado chega a ser de 16700 vezes mais rápido que o *A-Star* sequencial. Mostrando dessa forma, que o *A-Star* paralelizado tem uma vantagem muito grande com relação ao *A-Star* sequencial.

4
Conclusão

Após o levantamento dos pré-requisitos de hardware e parte da infraestrutura necessária, além de toda a fundamentação teórica no que tange ao entendimento do problema a ser solucionado por esse trabalho, foi realizado o desenvolvimento do roteamento do tráfego veicular através da teoria dos grafos aplicado ao algoritmo *A-Star*, que possui duas versões, uma sequencial e a outra paralelizada, que tiveram suas implementações validadas no cenário que abrange uma região do bairro Siqueira Campos em Aracaju.

Feita a validação dos algoritmos, realizou-se então uma análise comparativa dos dois algoritmos, observando o desempenho de ambos. A priori, foi possível constatar o coeficiente angular do processamento sequencial de 0,00198 enquanto que no caso paralelizado foi de 0,00089. Assim o tempo de processamento sequencial é por volta de 2,2 vezes mais demorado que o tempo de processamento utilizando o algoritmo CUDA.

Tendo em vista a primeira análise realizada e constatado que ambos os algoritmos tiveram bom desempenho, além do fato deles terem sido extressados com uma quantidades que variaram entre 100 mil a 1 milhão de veículos, supondo uma situação de anomalia uma vez que dificilmente uma região de proporções simuladas neste trabalho contará com tais quantidades de veículos.

Passado esse processo de validação e de *debug* dos códigos, os mesmos foram aplicados em simulações de tempo real na ferramenta de simulação SUMO, onde foram colocados para serem rodados calculando as melhores rotas em um cenários extraídos do mundo real, que foi o mapa de uma região do bairro Siqueira Campos da cidade de Aracaju - SE e também uma região de Manhattan - Nova Iorque - EUA. Mas agora, ambos sendo comparados com o algoritmo sequencial existente na própria ferramenta.

Com as execuções dos três algoritmos, um *A-Star* sequencial, outro *A-Star* paralelizado e o algoritmo sequencial do próprio SUMO, foram gerados vários resultados possibilitando uma realização de análise maior de cada um deles. Entre eles estão o tamanho médio das rotas

em metros, a duração média em segundos das viagens, o tempo total em milissegundos do processamento dos cálculos das rotas e por fim, o tempo médio também em milissegundos do processamento para os cálculos das rotas. Tudo isso, para ser tomada uma decisão de melhor algoritmo não baseado apenas em um parâmetro, mas sim a partir da análise da conjuntura.

Na análise do tamanho médio das rotas calculadas pelo algoritmo sequencial da própria ferramenta, percebeu-se que o mesmo tem grande vantagem em relação aos outros dois algoritmos, pois no final das execuções o mesmo teve um comportamento melhor que os demais algoritmos, quase chegando a ser o dobro de vantagem em relação ao algoritmo *A-Star* paralelizado. Em outras palavras, o algoritmo sequencial do SUMO calculou as rotas em metros muito menores que os outros.

A segunda métrica analisada foi o tempo médio em segundos das durações das viagens, pois o fato de que a rota seja a menor em metros, não implica que através dela o veículo consiga chegar mais rapidamente ao seu destino. Assim, ao verificar os resultados obtidos das execuções das simulações, pode-se afirmar que o algoritmo *A-Star* sequencial teve melhor desempenho comparado aos demais, visto que o mesmo foi sete vezes mais rápido que o algoritmo do SUMO.

Com isso, mesmo calculando rotas maiores em metros que o algoritmo da própria ferramenta, o algoritmo *A-Star* sequencial conseguiu fazer com que os veículos chegassem aos seus destinos mais rápido que os demais algoritmos analisados. Isto é, mesmo com rotas maiores em cenários onde as execuções são feitas em tempo real, pode-se obter rotas melhores onde o tráfego de veículos esteja menor, fazendo com que a viagem seja mais rápida em tempo de deslocamento.

Ainda assim, não se pode afirmar que seja o melhor algoritmo a ser escolhido mediante tais resultados, pois ainda não se sabe o quão demorado pode ser os cálculos realizados para encontrar ótimas rotas, tanto no que diz respeito a distância quanto as durações das viagens. A partir disso, houve a necessidade de se analisar dessa vez o tempo em que os cálculos estavam sendo realizados nos algoritmos *A-Star* sequencial quanto no *A-Star* paralelizado, devido o fato do algoritmo da ferramenta não fazer recálculos de rotas.

Haja vista os resultados obtidos das métricas de tempo total e médio em milissegundos de processamento dos cálculos, é possível observar a grande vantagem que se obteve ao utilizar o algoritmo *A-Star* paralelizado com relação ao algoritmo *A-Star* sequencial, chegando a ter um desempenho milhares de vezes melhor, realizando cálculos muito mais rápido tanto no tempo total quanto no tempo médio.

Mesmo não retornando rotas menores no sentido de distância, o algoritmo *A-Star* paralelizado compensa disponibilizando para os veículos rotas ótimas calculadas muito mais rapidamente, diminuindo drasticamente o tempo de espera. Em virtude disso, é válido ainda a sua implementação mesmo diante dos resultados apresentados na duração do tempo médio das viagens, onde o algoritmo *A-Star* paralelizado apresenta um diferença não muito relevante para

o *A-Star* sequencial que teve um comportamento melhor.

Os grandes centros urbanos tendem contar com um número ainda maior de habitantes, que acabam por sua vez necessitando de mais agilidade de locomação para atenderem as suas demandas, afetando dessa maneira significativamente de forma positiva na qualidade de vida da mesma ao terem o tempo reduzido de suas viagens, por contarem com um gerenciamento de tráfego veicular capaz de realizar cálculos para milhares de veículos em tempo muito hábil em ambientes cada vez mais dinâmicos.

Diante do exposto, o algoritmo *A-Star* paralelizado teve melhor comportamento comparado aos demais de forma geral, uma vez que o mesmo não tem muita desvantagem no tempo de duração de viagens, resaltando que neste caso ele não contou com todo o poder de processamento que a GPU disponibiliza, algo a ser levado em consideração já que a quantidade de veículos pode aumentar em determinados cenários e momentos, possibilitando dessa forma a utilização de mais *threads* da GPU tornando a realização dos cálculos nessa métrica mais eficientes.

Como trabalho futuro poderá ser feito uma melhoria no algoritmo *A-Star* paralelizado, no sentido de avaliar as heuristicas utilizadas pelo algoritmo da própria ferramenta SUMO para que se obtenha uma diminuição ou até mesmo superar a diferença do tamanho médio das rotas calculadas, melhorando ainda mais o desempenho do algoritmo *A-Star* paralelizado. Pois, além de contar com os cálculos realizados mais rapidamente ele poderá contar também com uma otimização nos cálculos das rotas no sentido das distâncias.

Referências

BACELLAR, H. V. Cluster: Computação de alto desempenho. *Instituto de Computação, Universidade Estadual de Campinas. Campinas, São Paulo*, 2009. Citado na página 21.

BARTH, M.; BORIBOONSOMSIN, K. Real-world carbon dioxide impacts of traffic congestion. *Transportation Research Record*, SAGE Publications Sage CA: Los Angeles, CA, v. 2058, n. 1, p. 163–171, 2008. Citado na página 10.

BEHRISCH, M. et al. Sumo – simulation of urban mobility: An overview. In: OMEROVIC, S. . U. of O. A.; SIMONI, R. I. R. T. P. D. A.; BOBASHEV, R. I. R. T. P. G. (Ed.). *SIMUL 2011*. ThinkMind, 2011. Disponível em: <https://elib.dlr.de/71460/>. Citado na página 15.

COSTA, C. A. Cidades inteligentes e big data. *Cidades inteligentes e mobilidade urbana. Cadernos FGV Projetos.*, v. 9, p. 66–73, 2014. Citado na página 10.

COSTA, C. A. Cidades inteligentes e big data. *Cidades inteligentes e mobilidade urbana. Cadernos FGV Projetos.*, v. 10, p. 120–121, 2015. Citado na página 10.

CRAUSER, A. et al. A parallelization of dijkstra's shortest path algorithm. In: SPRINGER. *International Symposium on Mathematical Foundations of Computer Science*. [S.l.], 1998. p. 722–731. Citado na página 24.

DENG, Y. et al. Fuzzy dijkstra algorithm for shortest path problem under uncertain environment. *Applied Soft Computing*, Elsevier, v. 12, n. 3, p. 1231–1237, 2012. Citado na página 25.

DETOMINI, R. C. Exploração de paralelismo em criptografia utilizando gpus. *SJR Preto, Universidade Julio de Mesquita Filho*, 2010. Citado 2 vezes nas páginas 18 e 19.

DUCHOŇ, F. et al. Path planning with modified a star algorithm for a mobile robot. *Procedia Engineering*, Elsevier, v. 96, p. 59–69, 2014. Citado na página 20.

FERGUSON, D.; LIKHACHEV, M.; STENTZ, A. A guide to heuristic-based path planning. In: *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*. [S.l.: s.n.], 2005. p. 9–18. Citado na página 19.

FERNANDES, R. M. A. XML y registros electrÃ: principales estÃ¡ndares en la descripciÃarchivÃstica. *CiÃda InformaÃ§Ã*, scielo, v. 35, p. 45 – 53, 12 2006. ISSN 0100-1965. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-19652006000300005&nrm=iso>. Citado na página 17.

Hart, P. E.; Nilsson, N. J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, v. 4, n. 2, p. 100–107, July 1968. ISSN 0536-1567. Citado na página 20.

HE, Z.; CAO, J.; LI, T. Mice: A real-time traffic estimation based vehicular path planning solution using vanets. In: IEEE. *2012 International Conference on Connected Vehicles and Expo (ICCVE)*. [S.l.], 2012. p. 172–178. Citado na página 12.

IBGE. *INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. Perfil dos municípios brasileiros 2013*. 2010. Disponível em: <ftp://ftp.ibge.gov.br/Perfil_Municipios/2013/munic2013.pdf>. Acesso em: 27 nov. 2018. Citado na página 10.

JASIKA, N. et al. Dijkstra's shortest path algorithm serial and parallel execution performance analysis. In: IEEE. *2012 proceedings of the 35th international convention MIPRO*. [S.l.], 2012. p. 1811–1815. Citado na página 24.

JOHNSON, T. T.; MITRA, S. Safe and stabilizing distributed multi-path cellular flows. *Theoretical Computer Science*, Elsevier, v. 579, p. 9–32, 2015. Citado na página 12.

KRAJZEWICZ, D. *Traffic simulation with SUMO–simulation of urban mobility*. [S.l.]: Springer, 2010. 269–293 p. Citado 2 vezes nas páginas 14 e 15.

LI, G.; HE, B.; DU, A. A traffic congestion aware vehicle-to-vehicle communication framework based on voronoi diagram and information granularity. *Peer-to-Peer Networking and Applications*, Springer, v. 11, n. 1, p. 124–138, 2018. Citado na página 12.

LIMA, J. C. de; CARVALHO, C. L. de. Extensible markup language (xml). 2005. Citado na página 17.

LUONG, T. V.; MELAB, N.; TALBI, E.-G. Gpu-based multi-start local search algorithms. In: *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*. Berlin, Heidelberg: Springer-Verlag, 2011. (LION'05), p. 321–335. ISBN 978-3-642-25565-6. Disponível em: <http://dx.doi.org/10.1007/978-3-642-25566-3_24>. Citado na página 19.

Naphade, M. et al. Smarter cities and their innovation challenges. *Computer*, v. 44, n. 6, p. 32–39, June 2011. ISSN 0018-9162. Citado na página 11.

NVIDIA. *NVIDIA CUDA programming guide*. 2009. Disponível em: <http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide_2.3.pdf>. Acesso em: 21 nov. 2018. Citado na página 19.

POSTSCAPES. *Anatomy of a Smart City*. 2015. Disponível em: <http://postscapes.com/anatomy-of-a-smart-city-full>. Acesso em: 2 dez. 2018. Citado 2 vezes nas páginas 10 e 11.

QIAN, J. et al. Accelerating reconfiguration for vlsi arrays with a-star algorithm. *IEEJ Transactions on Electrical and Electronic Engineering*, v. 13, n. 10, p. 1511–1519, 2018. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/tee.22716>. Citado na página 20.

RAUBER, T.; RÜNGER, G. *Parallel programming: For multicore and cluster systems*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 18.

SHIMOURA, H. et al. *Evaluation of the Effect of DRGS using Traffic Flow Simulation System*. [S.l.], 1995. Citado na página 11.

SIAU, n. z.; HINDE, C. J.; STONE, R. An evolution of a complete program using xml-based grammar definition. *IJCCI 2012 - Proceedings of the 4th International Joint Conference on Computational Intelligence*, 01 2012. Citado na página 17.

SOUSA, L. M. O.; ARAÚJO, E. M. d.; MIRANDA, J. G. V. Caracterização do acesso à assistência ao parto normal na bahia, brasil, a partir da teoria dos grafos. *Cadernos de Saúde Pública*, SciELO Public Health, v. 33, p. e00101616, 2017. Citado na página 19.

SUMO. *SUMO – Simulation of Urban MObility*. 2018. Disponível em: <https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/>. Acesso em: 3 dez. 2018. Citado na página 15.

SZWARCFITER, J. L. *Grafos e algoritimos computacionais*. [S.l.]: Campus, 1988. v. 2. Citado na página 19.

THULASIRAMAN, K.; SWAMY, M. N. *Graphs: theory and algorithms*. [S.l.]: John Wiley & Sons, 2011. Citado na página 19.

TOWNSEND, A. M. Book. *Smart cities : big data, civic hackers, and the quest for a new utopia / Anthony M. Townsend*. First edition. [S.l.]: W.W. Norton Company, Inc New York, NY, 2013. 120 – 132 p. ISBN 9780393082876 0393082873. Citado na página 11.

WEGENER, A. et al. Hovering data clouds: A decentralized and self-organizing information system. In: *Self-Organizing Systems*. [S.l.]: Springer, 2006. p. 243–247. Citado na página 12.

WEISS, M. C.; BERNARDES, R. C.; CONSONI, F. L. Cidades inteligentes: a aplicação das tecnologias de informação e comunicação para a gestão de centros urbanos. *Revista Tecnologia e Sociedade*, v. 9, n. 18, 2013. Citado 2 vezes nas páginas 10 e 11.

XU, Y. et al. Detecting urban road condition and disseminating traffic information by vanets. In: IEEE. *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. [S.l.], 2015. p. 93–98. Citado na página 13.